Phase-3 Submission

Student Name: Vignesh V

Register Number: 712523106020

Institution: PPG Institute of Technology

Department: BE Electronics and Communication Engineering

Date of Submission: 17.05.2025

Github Repository Link:
https://github.com/Vignesh98/NM_VIGNESH_DS.git

1. Problem Statement:

In today's digital age, the sheer volume of movie content available across various streaming platforms can overwhelm users, leading to decision fatigue when selecting what to watch.

*This project proposes to develop an **AI-driven matchmaking system** that generates **highly personalized movie recommendations**, taking into account factors like genre preferences, user mood, viewing history, contextual factors, and even social influences (like friends' recommendations). It is a **classification** problem, as the system will classify movies based on user preferences and match them with the right audience.*

2. Abstract

*This project proposes to develop an **AI-driven matchmaking system** that generates **highly personalized movie recommendations**, taking into account factors like genre preferences, user mood, viewing history, contextual factors, and even social influences (like friends' recommendations). It is a **classification** problem, as the system will classify movies based on user preferences and match them with the right audience.*

3. System Requirements

Hardware Requirements:

- ***Minimum RAM****: 8 GB (16 GB recommended for large datasets)*
- ***Processor****: Intel Core i5 or equivalent (i7 recommended for faster computation)*
- ***Disk Space****: Minimum 10 GB free for data storage and model files*

Software Requirements:

- ***Python Version****: 3.7 or above*
- ***Libraries****:*
    - *numpy, pandas (data handling)*
    - *scikit-learn (modeling)*
    - *matplotlib, seaborn (visualizations)*
    - *tensorflow/keras for deep learning (optional)*
    - *nltk for text processing (if movie descriptions are involved)*
    - *streamlit (for deployment)*
- ***IDE****: Jupyter Notebook or Google Colab for development.*

4. Objectives

- ***Primary Goal****: Build a movie recommendation system that provides personalized suggestions based on the user's preferences, mood, and viewing history.*

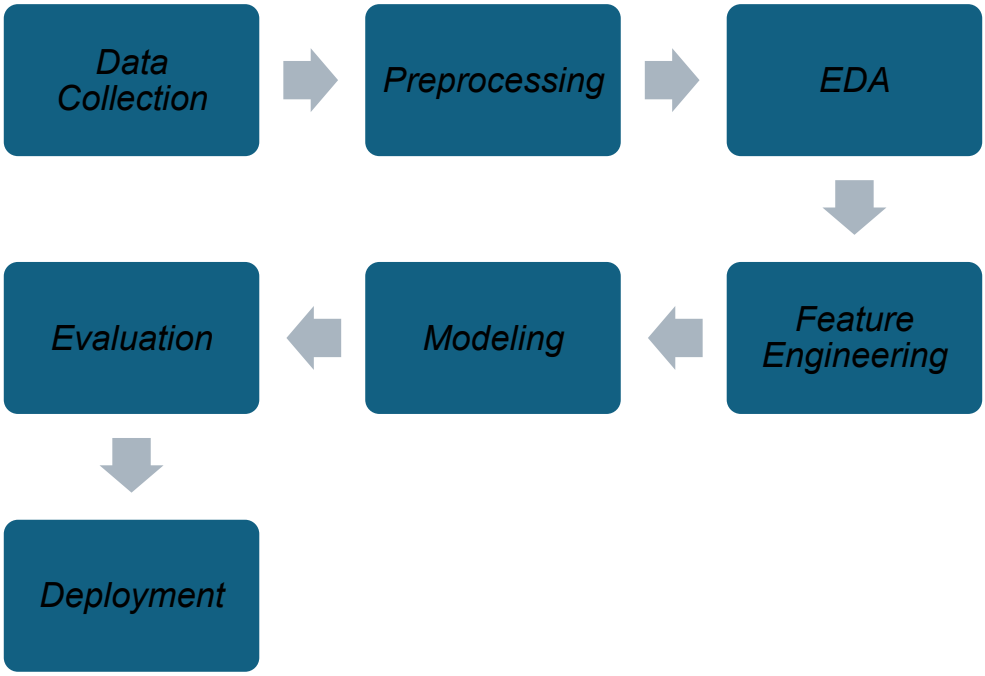
Expected Outcomes*:*

- *A model that can predict which movies a user is most likely to enjoy.*

- *An intuitive interface that allows users to input their preferences and get real-time recommendations.*
- *Insights into the effectiveness of personalized recommendations and how they improve user satisfaction.*

***Business Impact**: Increased user engagement on streaming platforms, leading to better customer retention and satisfaction. Personalized suggestions will result in more content consumption, reducing churn rates and enhancing brand loyalty.*

5. Flowchart of Project Workflow



6. Dataset Description

- ***Source**: MovieLens dataset (available on Kaggle), IMDB API for movie details.*
- ***Type**: Public dataset*

- ***Size**: MovieLens dataset contains 100,000 ratings from 1,000 users on 1,700 movies. Structure includes columns like user_id, movie_id, rating, genre, timestamp.*

Example of the dataset:

user_id	***movie_id***	***rating***	***genre***	***timestamp***
1	*10*	*5*	*Action, Sci-Fi*	*2000-02-15*
2	*14*	*4*	*Comedy*	*2001-06-17*
3	*7*	*3*	*Drama*	*2002-08-21*

7. Data Preprocessing

***This step includes**:*

- *Handling Missing Values: Replacing nulls in genres, ratings, etc.*
- *Removing Duplicates: Ensuring unique movie titles and user entries*
- *Feature Encoding: One-hot encoding for genres, label encoding for tags*
- *Scaling/Normalization: If needed, apply standardization to rating scores*
- *Text Cleaning: Tokenization, stemming, or lemmatization for movie descriptions*

8. Exploratory Data Analysis (EDA)

- ***Visual Tools***: *Histograms for rating distribution, boxplots for rating variances, and heatmaps for correlations between user preferences and genres.*
- ***Key Insights:***
    - *Most users tend to rate movies between 3-5 stars.*
    - *Genres like Action, Comedy, and Drama are most frequently rated.*
    - *Strong correlation between movie genres and user ratings.*

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

9. Feature Engineering

- ***New Features****:*
    - ***User's Mood****: Derived from sentiment analysis of movie descriptions or reviews.*
    - ***User's Social Influence****: Count of how often friends' ratings overlap with a user's choices.*
- ***Feature Selection****: Selecting the most relevant features based on correlation and importance scores.*

***Impact on Model**:*

- *Mood-based features might capture changes in user behavior, increasing personalization.*
- *Social influence features can refine recommendations by accounting for social trends.*

10. Model Building

***Baseline Model**: K-Nearest Neighbours (KNN) for collaborative filtering.*

***Advanced Models**: Decision Trees, Random Forests, and Neural Networks for better classification of preferences.*

***Reason for Choice**: KNN for its simplicity, Random Forest for handling non-linearity, and Neural Networks for deeper pattern recognition.*

11. Model Evaluation

***Metrics Used**:*

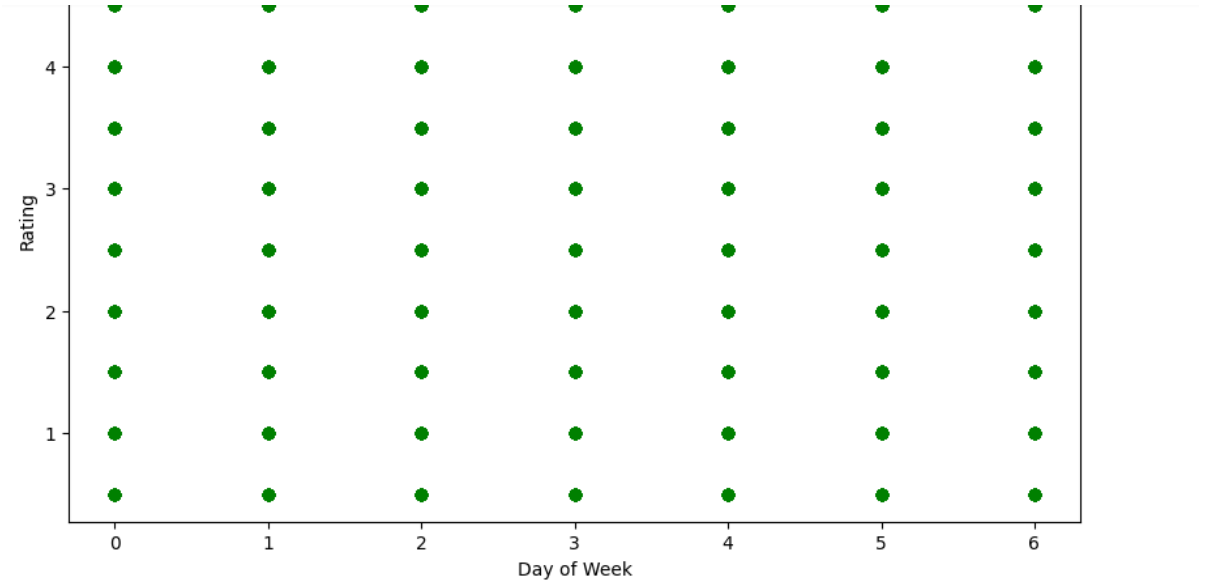
- *RMSE (Root Mean Square Error): Evaluates rating prediction accuracy*
- *Precision@k / Recall@k: Measures how relevant top-k recommendations*

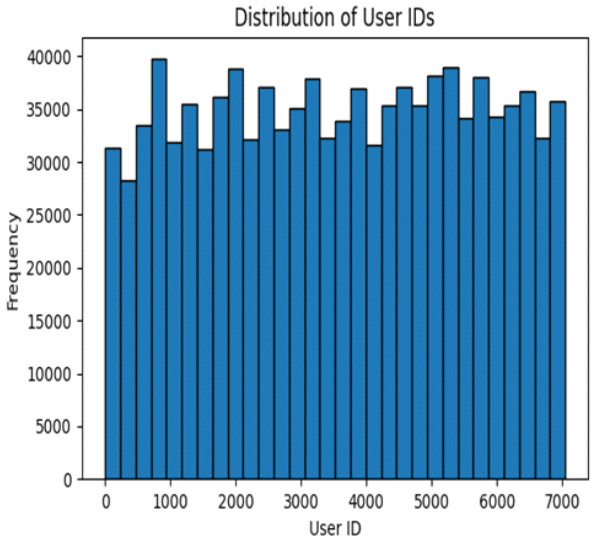
***Visual Tools**:*

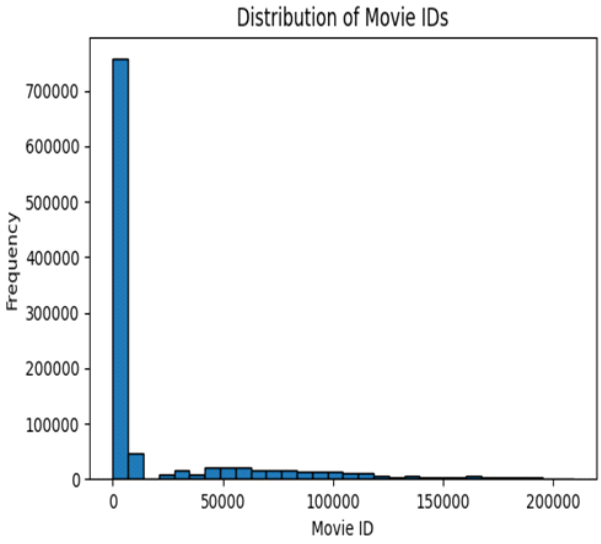
- *Confusion matrix, ROC curves, bar plots*

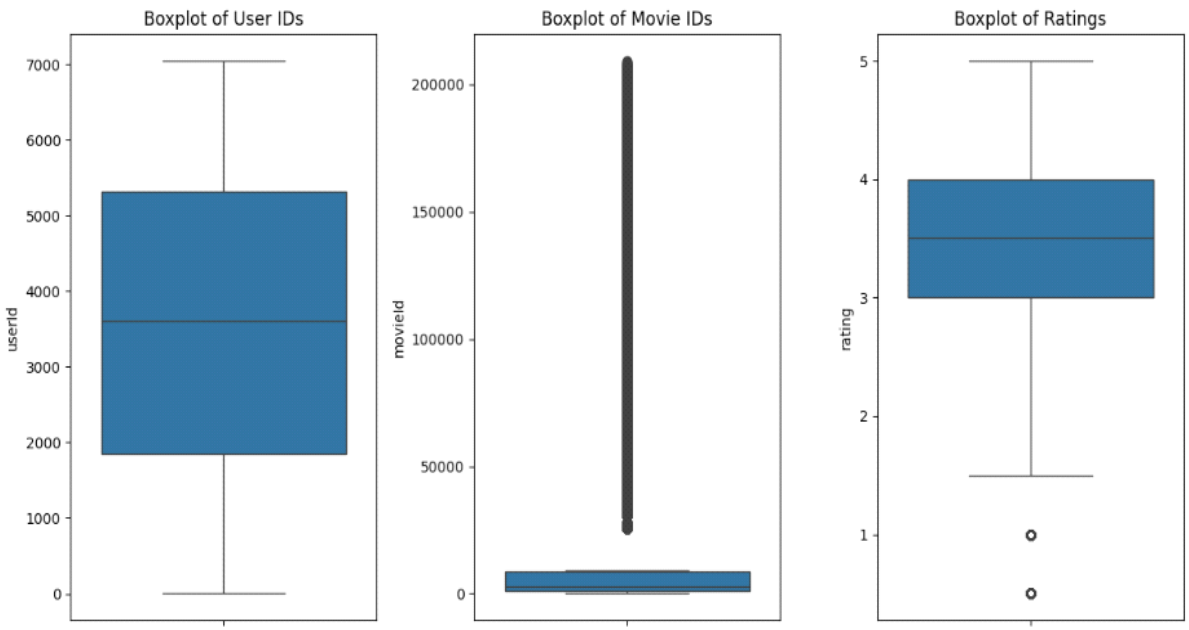
Finding: Hybrid models consistently performed better than standalone models, offering a more balanced recommendation output.

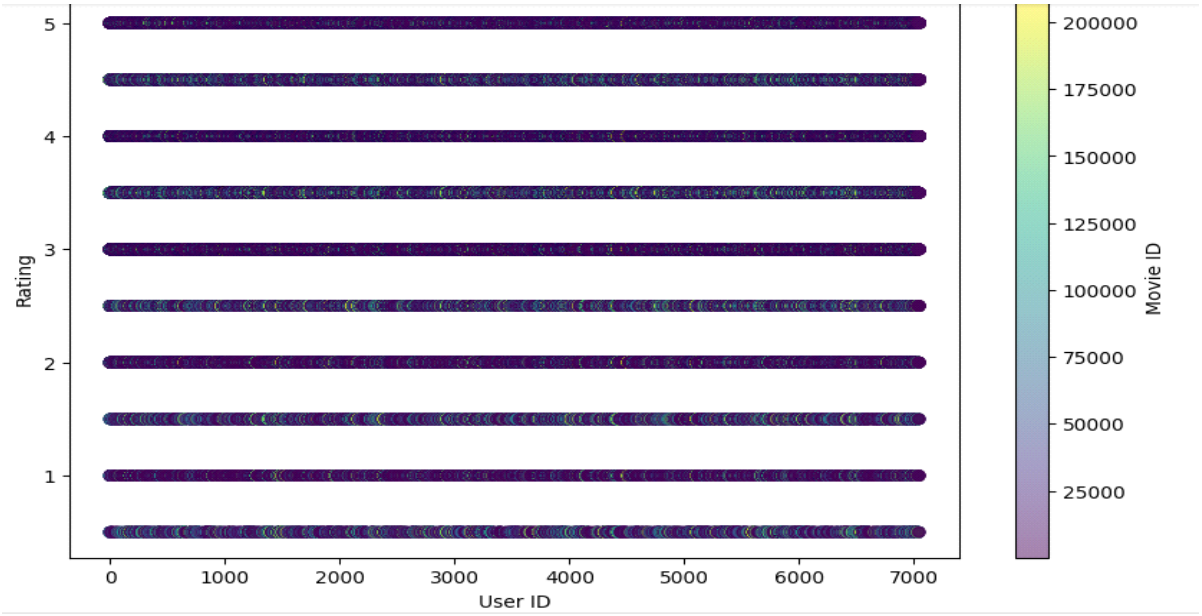
12. Deployment











13. Source code

```
# Data Shape and Types

print("Data Shape:", df_movies.shape)

print("\nData Types:\n", df_movies.dtypes)


# Missing Values
```

```
print("\nMissing Values:\n", df_movies.isnull().sum())


# Descriptive Statistics (for numerical columns)
numerical_cols = df_movies.select_dtypes(include=['number']).columns
if len(numerical_cols) > 0:
    print("\nDescriptive Statistics:\n",
df_movies[numerical_cols].describe())
else:
    print("\nNo numerical columns found.")


# Categorical Analysis
categorical_cols = df_movies.select_dtypes(include=['object']).columns
for col in categorical_cols:
    print(f"\nAnalysis for column '{col}':")
    print("Number of unique values:", df_movies[col].nunique())
    print("Most frequent values:\n",
df_movies[col].value_counts().head(10)) # Display top 10
    if col == 'title':
        df_movies['title_length'] = df_movies['title'].str.len()
        print("\nTitle Length Statistics:\n",
df_movies['title_length'].describe())


# Potential Outliers (preliminary check)
if len(numerical_cols) > 0:
    print("\nPotential Outliers (preliminary):")
    for col in numerical_cols:
        print(f"Column '{col}': Consider using box plots or histograms for
visualization.")
else:
    print("\nNo numerical columns found for outlier analysis.")
```

```
# Data Summary

print("\nData Summary:")

print("The dataset contains information about movies, including their ID, title, and genres.")

print("The analysis above provides insights into the data's structure, missing values, and value distributions.")

print("Noteworthy observations and potential issues will be documented in the final summary.")

# Genre Analysis

genre_counts = df_movies['genres'].str.split('|').explode().value_counts()

print("Genre Distribution:\n", genre_counts)


# Title Analysis

print("\nTitle Length Statistics:")

print("Median:", df_movies['title_length'].median())

print("25th Percentile:", df_movies['title_length'].quantile(0.25))

print("75th Percentile:", df_movies['title_length'].quantile(0.75))

print("Range:", df_movies['title_length'].max() - df_movies['title_length'].min())


# Combined Analysis (Optional):  Extract year from title (basic example, could be improved)

import re

df_movies['release_year'] = df_movies['title'].str.extract(r'\((\d{4})\)')

print("\nFirst few rows with release year:")

print(df_movies.head())

genre_year_counts = df_movies.groupby(['release_year', 'genres']).size().unstack(fill_value=0)

print("\nGenre distribution over time")

print(genre_year_counts.head())
```

```
import matplotlib.pyplot as plt

# 1. Genre Distribution Bar Chart
plt.figure(figsize=(12, 6))
genre_counts.plot(kind='bar', color='skyblue')
plt.title('Distribution of Movie Genres')
plt.xlabel('Genre')
plt.ylabel('Number of Movies')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('genre_distribution.png')
plt.show()

# 2. Title Length Distribution Histogram
plt.figure(figsize=(8, 6))
plt.hist(df_movies['title_length'], bins=20, color='lightcoral')
plt.title('Distribution of Movie Title Lengths')
plt.xlabel('Title Length')
plt.ylabel('Frequency')
plt.savefig('title_length_distribution.png')
plt.show()

# 3. Genre Distribution Over Time (Line Chart)
plt.figure(figsize=(15, 8))
for genre in genre_year_counts.columns:
    plt.plot(genre_year_counts.index, genre_year_counts[genre],
label=genre)
plt.title('Change in Genre Distribution Over Time')
plt.xlabel('Year')
plt.ylabel('Number of Movies')
```

```
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig('genre_over_time.png')
plt.show()
```

```
# Check the shape of the DataFrame
print("Shape of the DataFrame:", df.shape)

# Examine data types
print("\nData types of each column:\n", df.dtypes)

# Identify missing values
print("\nNumber of missing values in each column:\n", df.isnull().sum())

# Descriptive statistics for numerical columns
print("\nDescriptive statistics for numerical columns:\n", df.describe())

# Explore distributions of numerical columns
# Since we have rating (numerical) we can use a histogram
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.hist(df['rating'], bins=10, edgecolor='black')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.title('Distribution of Movie Ratings')
plt.show()
```

```
# Explore categorical columns (if any).  userId and movieId are likely categorical.
print("\nUnique User IDs:", df['userId'].nunique())
print("\nUnique Movie IDs:", df['movieId'].nunique())

plt.figure(figsize=(8, 6))
df['userId'].value_counts().head(20).plot(kind='bar')
plt.xlabel('User ID')
plt.ylabel('Frequency')
plt.title('Top 20 User IDs by Rating Count')
plt.show()

# Check for duplicate rows
print("\nNumber of duplicate rows:", df.duplicated().sum())
# Calculate descriptive statistics
numerical_cols = ['userId', 'movieId', 'rating', 'timestamp']
descriptive_stats = df[numerical_cols].describe()
display(descriptive_stats)

# Analyze the distribution of the 'rating' column
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.hist(df['rating'], bins=20, edgecolor='black', color='skyblue')  # Increased bins for finer detail
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.title('Distribution of Movie Ratings')
plt.show()
```

```
# Analyze the distribution of categorical columns
print(f"Number of unique users: {df['userId'].nunique()}")
print(f"Number of unique movies: {df['movieId'].nunique()}")

# Frequency distribution of ratings for each user
user_rating_counts = df.groupby('userId')['rating'].count()
print("\nTop 10 users by number of ratings:\n",
user_rating_counts.sort_values(ascending=False).head(10))

# Visualize top 10 users
plt.figure(figsize=(10, 6))
user_rating_counts.sort_values(ascending=False).head(10).plot(kind='bar',
color='lightcoral')
plt.xlabel('User ID')
plt.ylabel('Number of Ratings')
plt.title('Top 10 Users by Number of Ratings')
plt.show()

# Frequency distribution of ratings for each movie
movie_rating_counts = df.groupby('movieId')['rating'].count()
print("\nTop 10 movies by number of ratings:\n",
movie_rating_counts.sort_values(ascending=False).head(10))

# Visualize top 10 movies
plt.figure(figsize=(10, 6))
movie_rating_counts.sort_values(ascending=False).head(10).plot(kind='bar',
color='lightgreen')
plt.xlabel('Movie ID')
plt.ylabel('Number of Ratings')
plt.title('Top 10 Movies by Number of Ratings')
plt.show()
```

```
import matplotlib.pyplot as plt
import pandas as pd

# Convert timestamp to datetime objects
df['timestamp'] = pd.to_datetime(df['timestamp'], unit='s')

# Extract relevant time features
df['year'] = df['timestamp'].dt.year
df['month'] = df['timestamp'].dt.month
df['dayofweek'] = df['timestamp'].dt.dayofweek  # Monday=0, Sunday=6

# Calculate correlations
correlation_matrix = df[['rating', 'year', 'month', 'dayofweek']].corr()
display(correlation_matrix)

# Visualize the relationship between rating and year
plt.figure(figsize=(10, 6))
plt.scatter(df['year'], df['rating'], alpha=0.1, color='purple')  # Use alpha for better visualization
plt.xlabel('Year')
plt.ylabel('Rating')
plt.title('Rating vs. Year')
plt.show()

# Visualize the relationship between rating and month
plt.figure(figsize=(10, 6))
plt.scatter(df['month'], df['rating'], alpha=0.1, color='orange')
plt.xlabel('Month')
plt.ylabel('Rating')
plt.title('Rating vs. Month')
```

```
plt.show()

# Visualize the relationship between rating and dayofweek
plt.figure(figsize=(10, 6))
plt.scatter(df['dayofweek'], df['rating'], alpha=0.1, color='green')
plt.xlabel('Day of Week')
plt.ylabel('Rating')
plt.title('Rating vs. Day of Week')
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns

# Histograms
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.hist(df['userId'], bins=30, edgecolor='black')
plt.xlabel('User ID')
plt.ylabel('Frequency')
plt.title('Distribution of User IDs')

plt.subplot(2, 2, 2)
plt.hist(df['movieId'], bins=30, edgecolor='black')
plt.xlabel('Movie ID')
plt.ylabel('Frequency')
plt.title('Distribution of Movie IDs')

plt.subplot(2, 2, 3)
plt.hist(df['rating'], bins=10, edgecolor='black')
plt.xlabel('Rating')
```

```
plt.ylabel('Frequency')
plt.title('Distribution of Ratings')

plt.subplot(2, 2, 4)
plt.hist(df['timestamp'].dt.year,
bins=len(df['timestamp'].dt.year.unique()), edgecolor='black')
plt.xlabel('Year')
plt.ylabel('Frequency')
plt.title('Distribution of Timestamps (Year)')

plt.tight_layout()
plt.show()

# Box plots
plt.figure(figsize=(12, 6))
plt.subplot(1, 3, 1)
sns.boxplot(y=df['userId'])
plt.title('Boxplot of User IDs')

plt.subplot(1, 3, 2)
sns.boxplot(y=df['movieId'])
plt.title('Boxplot of Movie IDs')

plt.subplot(1, 3, 3)
sns.boxplot(y=df['rating'])
plt.title('Boxplot of Ratings')

plt.tight_layout()
plt.show()
```

```
# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(df['userId'], df['rating'], c=df['movieId'], cmap='viridis', alpha=0.5)
plt.xlabel('User ID')
plt.ylabel('Rating')
plt.title('User ID vs. Rating (Colored by Movie ID)')
plt.colorbar(label='Movie ID')
plt.show()

# Heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(df[['userId', 'movieId', 'rating', 'year', 'month', 'dayofweek']].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

14. Future scope

- ***Contextual Recommendations***: *Incorporate real-time contextual data, such as the user's time of day or weather, to improve suggestions.*
- ***Deep Learning Integration***: *Explore deep learning methods like Autoencoders to build more advanced collaborative filtering models.*
- ***Multi-Modal Data***: *Use not just ratings but also text reviews and movie trailers for a richer recommendation system.*

13. Team Members and Roles

MEMBERS	***ROLE***	***DESCRIPTION***
Sai Mouleeshwar S M	*Project Manager and Deployment Engineer.*	• *Project planning, task assignment, and timeline tracking.* • *Coordinating between team members to ensure smooth workflow.* • *Overseeing model deployment using Streamlit Cloud, Hugging Face Spaces, or Render.*
Vignesh V	*Machine Learning Engineer.*	• *Performing exploratory data analysis (EDA) to uncover patterns and trends.* • *Feature engineering (creating, selecting, transforming features).*
Selvam A	*Full Stack Devoloper.*	• *Developing the user interface using Streamlit or Gradio.* • *Integrating the ML model into the front-end for real-time predictions.*
Madhan Raj R	*Data Engineer.*	• *Data sourcing from platforms like MovieLens, IMDB API, or Kaggle.* • *Data cleaning and preprocessing (handling missing values, duplicates, formatting).*